Deferred Vector Map Visualization

Thöny, Matthias ; Billeter, Markus ; Pajarola, R

Abstract: Interactive rendering of large scale vector maps is a key challenge for high-quality geographic visualization software systems. In this paper we present a novel approach for the visualization of large scale vector maps over detailed height-field terrains. Our method uses a deferred line shading approach to render large scale vector maps directly in a screen-space shading stage over a terrain visualization. The fact that there is no traditional geometric polygonal rendering involved allows our algorithm to outperform conventional vector map rendering algorithms for geographic information systems. Our flexible clustered deferred line rendering approach allows a user to interactively customize and apply advanced vector styling methods, as well as the integration into a vector map level-of-detail system.

DOI: https://doi.org/10.1145/3002151.3002157

Deferred Vector Map Visualization

Matthias Thöny* Markus Billeter* Renato Pajarola*
Departement of Informatics, University of Zürich

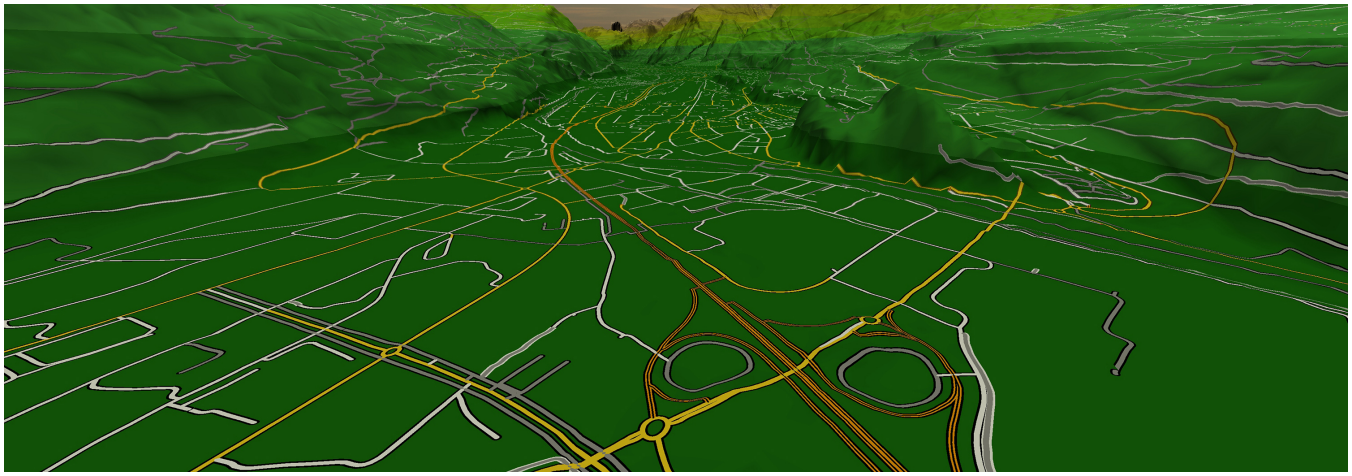


Figure 1: *Example of a large scale vector map data set showing a part of a street network consisting of 16 million line segments.*

Abstract

Interactive rendering of large scale vector maps is a key challenge for high-quality geographic visualization software systems. In this paper we present a novel approach for the visualization of large scale vector maps over detailed height-field terrains. Our method uses a deferred line shading approach to render large scale vector maps directly in a screen-space shading stage over a terrain visualization. The fact that there is no traditional geometric polygonal rendering involved allows our algorithm to outperform conventional vector map rendering algorithms for geographic information systems. Our flexible clustered deferred line rendering approach allows a user to interactively customize and apply advanced vector styling methods, as well as the integration into a vector map level-of-detail system.

Keywords: geographic visualization, vector map, line rendering

Concepts: •**Human-centered computing** → **Geographic visualization;** •**Computing methodologies** → *Rendering;*

1 Introduction

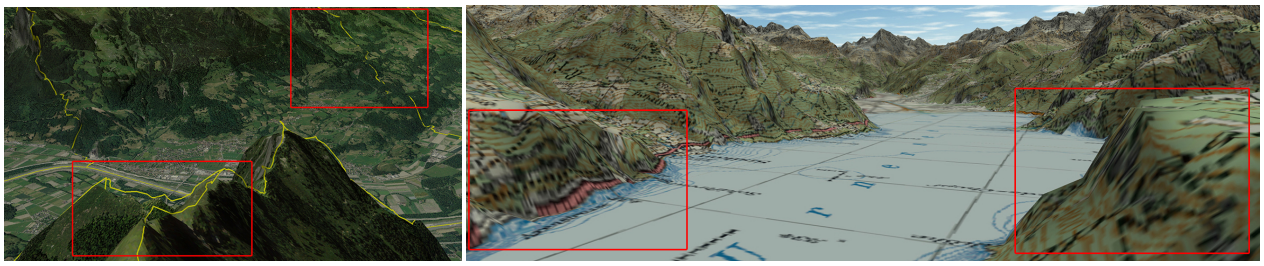
Vector map visualizations are often part of geographical information systems such as virtual globe software, mapping and navigation systems or other geo-spatial real-time 3D environments. Improving the interactive visualization of vector maps will allow these software products to show and interact with more data and in a more precise way. Vector maps are used to represent geographic features such as streets, rivers, contour lines or land use information. An example of such data can be seen in Fig. 1. Displaying and visualizing these data sets interactively in real-time 3D applications, such as Google Earth, Caesium Virtual Globe and Map Engine or NASA Worldwind is a challenging task. In this paper we focus on the problem of rendering large scale vector maps with many millions of line segments within a 3D real-time geo-visualization application. The massive amount of vector map data is challenging because of limited memory capacities and because of the rendering time per frame which should be minimized for real-time purposes.

In geographic visualization systems, previous vector map rendering methods may cause visual artifacts that negatively affect the interactive data exploration quality and corresponding geo-spatial analysis tasks. Examples of such artifacts are shown in Fig. 2. In particular, when combining multiple vector maps and a continuous multiresolution level-of-detail terrain mesh the mismatching resolutions cause significant artifacts in visualization. The problems in Fig. 2(a) occur because line segments of differing resolution vector maps (in yellow) float above or intersect the terrain. This unpredictable scene configuration makes the problem more complicated for geometric line rendering methods. Texture based approaches can solve the intersection problem, but suffer from other artifacts such as aliasing and projective distortions as shown in Fig. 2(b).

Compared to other vector map rendering methods, our approach performs a deferred shading pass for the vector map data on top of the traditionally rendered terrain surface such as to avoid dependence on modified (preprocessed) vector map geometry. Consequently this avoids the coupling between different vector maps or between vector maps and the terrain height-field during a preprocessing stage, and all data set combinations are changeable at runtime. Furthermore, we can avoid on one hand artifacts as shown in Fig. 2, and on the other hand we also achieve pixel precise line display results even with multiple layers of vector maps with different level-of-detail resolutions being visualized on top of the terrain. Moreover, modern map visualization systems should allow the user to interactively change their map visualization with advanced vec-

*e-mail:{mthoeny|billeter|pajarola@ifi.uzh.ch}

SA '16 Symposium on Visualization , December 05-08, 2016, , Macao
ISBN: 978-1-4503-4547-7/16/12
DOI: http://dx.doi.org/10.1145/3002151.3002157



(a) (b)

Figure 2: *Example artifacts when rendering vector maps. (a) Vector lines floating or intersecting the underlying 3D terrain, and (b) texture mapping based projection and resolution artifacts.*

tor styling methods. This requires a flexible line rendering method capable of extending the geometric line rendering to a line styling display concept. In this paper we take these requirements into account and allow interactive modification of vector maps on top of the terrain visualization.

To achieve fast rendering, we will show how a deferred rendering approach can be exploited to interactively visualize large scale vector maps with many millions of line features. In particular, we demonstrate how a clustered spatial line segment hierarchy can improve our deferred line rendering pipeline by optimizing the GPU workload for every pixel.

2 Related Work

In the following section we review state-of-the-art techniques for vector map rendering, as well as relevant work related to clustered deferred shading. The terrain rendering in this work is based on RASTeR [Bösch et al. 2009; Goswami et al. 2010] but could be replaced by other state-of-the-art systems such as [Losasso and Hoppe 2004; Dick et al. 2009; Livny et al. 2009; Ripolles et al. 2012; Kang et al. 2015].

2.1 Vector Map Visualization

Vector maps as shown in Figs. 1 and 2 are usually line or polygon based data describing geometric objects with specific attributes. This geometric information can be used directly for a 3D visualization as shown in [Bruneton and Neyret 2008]. However, the most popular method is the combination of vector maps with image based information, like aerial photographic data, projected onto a terrain height field surface model. The closest related approaches for vector map visualizations discussed below can be divided into three categories: (1) texture based, (2) applying geometric subdivision and (3) using shadow volumes. These methods are described in more detail in the survey of interactive visualization of vector data [Kersting and Döllner 2002] and the survey of a digital earth [Mahdavi-Amiri et al. 2015]. A possible system description for these methods can be found in [Cozzi and Ring 2011]. In Tab. 1 we summarize the main advantages and limitations of these three approaches in comparison to our new deferred vector map visualization method.

A common method used in geo-visualization systems is the texture based approach of rendering vector maps. Different descriptions of this method as well as comparisons to other methods can be found in [Kersting and Döllner 2002; Wartell et al. 2003; Sun et al. 2008; Wang et al. 2009]. The basic idea is that vector maps are (orthogonally) rasterized to images and used as textures mapped on the terrain, e.g. as in Fig. 2(b). In general, any rendering system can easily apply textures to (terrain) surfaces, therefore, it is convenient and simple to implement such a texture based vector map visualization. In addition, texture based methods are often used if the development targets have limited hardware capabilities such as embedded devices, mobile platforms or browser based applications.

Texture based vector maps, however, may suffer from artifacts as shown in Fig. 2(b). The highlighted artifacts are caused by the 2D texture projection as well as stem from the limited texture resolution. If the texture resolution is increased, however, more memory is needed. Many systems thus work with preprocessed texture pyramids, but adding higher resolutions increases on one hand the memory consumption and on the other hand also the amount of texture files in these systems exponentially. Additionally, the texture pyramid may introduce border artifacts of the vector map information during rendering when used in a multiresolution level-of-detail system. Furthermore, most of the systems using this type of visualization do not allow immediate modification and styling of vector maps within an interactive 3D display session. To achieve a precise and suitable visualization for interactive vector map modification, it is necessary to manage a dynamic texture pyramid and to implement an on-the-fly rasterization step for updating vector maps. The amount of re-rasterization grows with the complexity of modification possibilities such as selection of vector map layers or highlighting of selected elements. Artifacts often appear when moving the view frustum close to the surface and the camera points to a far distance. In these cases, the texture based approach can get overly complex and costly in terms of memory, rendering time and system flexibility.

In geometric subdivision approaches for vector maps [Kersting and Döllner 2002; Schneider et al. 2005; Xu et al. 2010; Deng et al. 2013] lines are subdivided according to the terrain mesh structure as illustrated in Fig. 3. Along the line segment at every change in slope of the underlying terrain, corresponding to crossing triangle edges, the line segment is subdivided. An application of the geometric line subdivision in combination with advanced map styling features can be found in [Vaaraniemi et al. 2011] and [Wilkie et al. 2012]. The methods show possible ways to do precise line renderings. However, this geometric approach for vector maps has the drawback that its line subdivision requires a predetermined and fixed combination of terrain triangulation and vector map subdivision. Dynamically changing terrain information, as is the case in continuous LOD terrain rendering, as well as unforeseeable combinations or editing of vector map data sets are hard to manage in this approach, and therefore, interaction possibilities are limited. This problem becomes even harder when the terrain information contains multiple layers. This is the case when height maps of different resolutions are combined and transitions between them are generated dynamically. It cannot be assumed anymore that a single

Criteria	Geometric Approach	Texture Mapping Approach	Shadow Volumes	Deferred Vector Maps
Rendering artifacts (Fig. 2)	Intersections, z-Buffer artifacts	texture aliasing, distortions	geometric aliasing	geometric aliasing
Output accuracy	tesselation resolution	texture resolution	pixel-accurate	pixel-accurate
Dynamic changes	recompute tesselation	redraw textures	yes*	yes
Interactive styling & editing	recompute tesselation	evaluate texture content	missing geometry at shading	update line buffer
Memory consumption	geometry only	hi-res textures	geometry only	geometry only
Additional geometry needed	recompute tesselation	none	geometry extrusion	only line information
Preprocessing requirements	complete terrain necessary	texture hierachy	no preprocessing	line buffer generation
GPU requirements	none	none	geometry shader*	random memory reads
Applicable to large data	intensive preprocessing	expensive redraw/preprocessed	pixel overdraw too expensive	demonstrated with $> 10^6$ lines

Table 1: *The table summarizes the differences between previous methods and our new technique (last column). *GPU requirements may be traded for a preprocessing step, making dynamic changes more costly.*

point has a unique fixed height value, and often terrain blending is done in the shader stage such that the mesh cannot be retrieved. In such cases graphical artifacts would appear. Another aspect is the precise overlay of planar geometric objects, which often leads to z-buffer artifacts (*z-fighting*) due to the limited precision of the depth buffer.

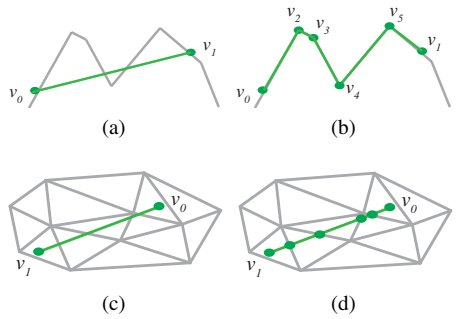


Figure 3: *Example for subdivision of a line according to the terrain height field. (a) Shows the elevation profile and (b) the subdivision. (c) Shows the line on top of the terrain and (d) the subdivision of the line matching the terrain mesh.*

The shadow volume approach for vector map visualizations produces high quality solutions because the algorithm works independent from the terrain resolution and always produces a pixel precise result on the screen [Dai et al. 2008; Wang et al. 2009; Yang et al. 2011]. The idea is that the geometry of a vector map is orthographically projected on the terrain by extruding the vector map's line segments into 3D polyhedral objects. The vertically extruded polyhedrons are then rendered in two steps, first front faces then back faces. Analog to shadow volumes, every screen pixel counts the difference between front and back faces. The result per pixel then contains the information if this pixel is a part of a projected vector map or not. The main drawback of this method is that multiple geometry rendering passes are required for the vector map, in addition to the terrain, and that the vector map geometry has to be extruded, e.g. in a geometry shader. On one hand the amount of geometry is thus about four times bigger than in the original vector map, and on the other hand every extruded line segment has to be rendered twice. Furthermore, in case of large vector maps the vertically extruded geometry covers large portions of the screen and may produce a massive overdraw. Overall, this severely limits the technique to rendering very moderately sized vector maps of maybe a few thousand line segments. A special case is shown in [Ohlarik and Cozzi 2011] where the method is optimized for vector maps only containing lines. Furthermore, it is hard to preserve the original vector map information so that advanced vector styling or procedural texturing can be achieved.

2.2 Deferred Shading

Our clustered deferred line rendering approach is inspired by the way lighting calculations are done in deferred shading pipelines [Saito and Takahashi 1990; Liktor and Dachsbacher 2012]. Deferred shading is applied to reduce the amount of shading operations by introducing a two-pass rendering pipeline. The first pass renders the geometry and produces a set of textures containing geometric scene information such as color, normal, depth or light information. The collection of output textures of the first pass is called a G-buffer, see also Fig. 4. All shading operations are done as image based effects using the information from the G-buffer in as few shading passes as possible. These render passes implement the effective shading and lighting for every pixel as well as other screen-space post processing operations such as e.g. antialiasing [Chajdas et al. 2011] or ambient occlusion [Bavoil and Sainz 2008]. Building up a deferred shading pipeline raises on one hand the GPU memory consumption for additional texture layers, and on the other hand the pixel fill rate because many more images are produced than effectively used as final output frames. Eventually the overall rendering effort per frame can nevertheless be reduced drastically and allows for more image-space effects within one single frame. Clustered Deferred Shading described in [Olsson et al. 2012] subdivides the view frustum into clusters to improve the rendering speed for scenes with many lights. In our concept we took over the idea of clustering scene objects by creating line clusters in world-space as a preprocess.

3 Deferred Vector Map Rendering

All approaches discussed above are using some kind of extracted geometry or geometry rasterized on textures for vector map visualizations. However, modern programmable GPUs allow us to develop much more flexible systems. The basic idea of our line projection and rendering approach is to directly project and display vector maps on top of the terrain surface using an adaptation of the deferred shading principle without the need to generate intermediate geometric objects or textures. In contrast to common rendering methods where the color of a pixel is derived from the main (geometry) rendering pass, our approach inverts this principle for vector map visualization.

In Fig. 4 we outline the main steps of our deferred line rendering method as further detailed below. In a deferred shading stage, for every pixel corresponding to a point on the terrain it is determined if it contributes to the visualization of a vector map feature. Using the G-buffer data obtained from the main terrain rendering pass, we back-project each pixel into the 3D world and determine its location within the vector map, see also Fig. 6. To identify candidate line features, we use a clustered buffer storing all vector map line elements, which we call a *clustered line buffer*. An optimized search

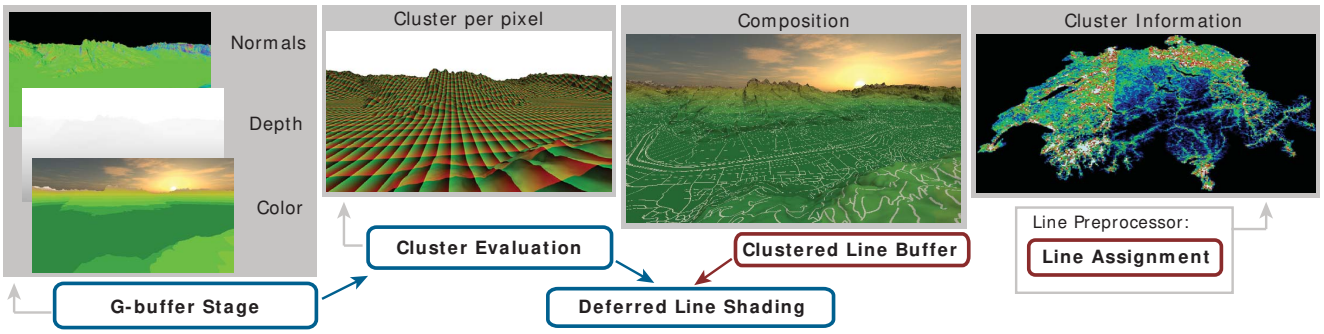


Figure 4: *Our deferred line rendering pipeline for large scale vector map visualization. After the generation of the G-buffer, the cluster evaluation is performed and used as input in combination with the clustered line buffer for the deferred vector map line shading. The clustered line buffer is prepared in the line assignment preprocess.*

within the clustered line buffer clusters allows us to find the closest line feature quickly and color pixels accordingly.

3.1 Clustered Line Buffer

During the deferred shading stage, for each pixel the closest intersecting line feature, if any, must be determined very quickly. For this we use our *clustered line buffer* which *clusters* the individual line segments of the vector map features into a large 2D structure of cells. Every line segment is assigned to each buffer cell it intersects. This can be a regular grid as currently implemented, but it could also be a multi-level nested grid or other space-partitioning hierarchy to better adapt to variations in the feature density in very large vector maps. Important is the ability to perform point-to-cell look-ups very efficiently, to allow fast identification of the cluster containing the closest lines potentially intersecting a back-projected pixel. Moreover, within each cluster the line segments are organized in an effective spatial search index structure to accelerate line search and pruning as well as minimize distance calculations after the coarse cluster identification.

The clustered line buffer is thus a data structure enabling fast point-to-line search queries and is designed to be used efficiently on the GPU during the deferred shading pass, as illustrated in Fig. 5(a). The clusters of this line buffer are formed during a preprocessing pass which assigns all vector map line segments to their corresponding clusters as further described below.

Two *line segment buffer* arrays store the start- and end-point coordinates $\{p_s, p_e\}$ of the individual line segments l_i on the GPU. A single *line segment index buffer* stores the indices to line segments concatenated for all cluster. The cluster grid is represented by two 2D integer textures, $o_{u,v}$ representing the cluster's *index offset* into the line segment index buffer and $c_{u,v}$ for the *index count* denoting the number of lines in the cluster. In a line assignment preprocess, for each cluster $C_{u,v}$ the vector map line features intersecting it are recorded, counted, and then concatenated to form the line segment index buffer.

The texture sizes for $o_{u,v}$ and $c_{u,v}$ equal the size of the cluster grid $C_{u,v}$. On one hand the grid should not be too coarse, because that would include too many line segments within each cluster. On the other hand the amount of clusters is limited to the texture memory that can be committed. In our implementation we typically divided the map space into 256×256 clusters to express the cluster indices u, v as one byte each. Other multi-level nested grids or space partitioning structures could be used as well, given a memory efficient implementation and fast cluster identification on the GPU.

The line assignment to a specific cluster follows a Bresenham line rasterization pattern as illustrated in Fig. 5(b), with certain line segments being assigned to several clusters. As the line drawing style is not known beforehand, we currently assume a predetermined conservative maximal line width which is incorporated into the line assignment. As indicated in Fig. 5(b), the line segment $l_1 = \{p_1, p_2\}$ lies in at least two clusters, but since the line has a certain line width we also have to assign its line segment index to all clusters it overlaps, e.g. $C_{1,2}$ too. Similar for $l_2 = \{p_3, p_4\}$ all overlapping clusters along the line are included, e.g. including also $C_{3,1}$.

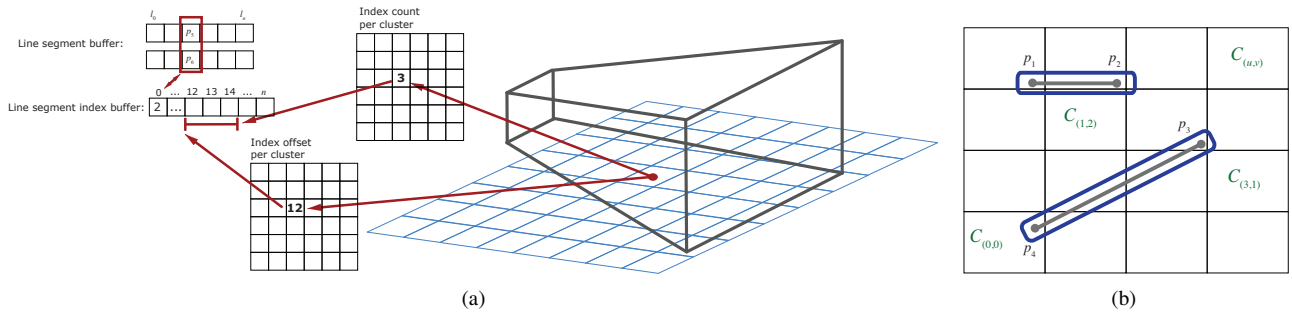
To optimize the point-to-line query after the coarse point-to-cluster location, we use a hierarchical spatial index structure to organize all line segments within one cluster $C_{u,v}$. For simplicity combined with efficiency we generate a fully balanced binary bounding volume hierarchy (BVH). In the preprocess we first order all lines within each cluster based on their midpoint along a space filling curve. Then we build a binary tree bottom-up forming a balanced BVH. Other optimized BVH construction approaches could further improve upon this solution.

3.2 Deferred Line Shading

The final fragment color for a screen pixel is determined in the *deferred line shading* stage, see also Fig. 4. In this stage our approach decides whether a pixel contributes to the visualization of a vector map element or not. Using the clustered line buffer, where all line segments are grouped into clusters, the search space of all line segments influencing a single pixel can be restricted to the line segments belonging to a certain cluster into which the pixel projects. Hence the line identification consists of two main steps: in the first step it is necessary to determine the cluster in the clustered line buffer affecting a certain pixel. The second step determines if and which line effectively intersects the given pixel considering the applied line-style width.

The cluster of a certain pixel $s_{(x,y)}$ is determined by a backwards projection of the pixel position from screen space to world space coordinates. For every screen position $s_{(x,y)}$ a back projection can be applied using the corresponding depth $d_{(x,y)}$ from the G-buffer information, illustrated in Fig. 6 by the red line. The back projection of $s_{(x,y,d_{(x,y)})}$ then results in the point I_s which is the pixel's location in world coordinates. The back projection can be expressed as multiplication with the inverse view-projection matrix M_{VP} as $I_s = M_{\mathrm{VP}}^{-1} \cdot s_{(x,y,d_{(x,y)})}$.

With the information about the clustered line buffer's subdivision of the vector map and the point I_s in world space, it is easy to cal-



(a) (b)

Figure 5: *(a) The* clustered line buffer *GPU data structures consist of multiple array buffers and textures supporting efficient point-to-cluster location and point-to-line search queries from a given geographical location. (b) Assignment of feature lines to clusters by rasterization.*

culate the cluster $C_{u,v}$ containing the point I_s. In other words, this step maps the screen-space pixel locations (x,y) to the cluster-index space (u,v). In Fig. 4 we visualize this cluster evaluation by coloring each pixel in (red,green) based on its relative position within the corresponding cluster.

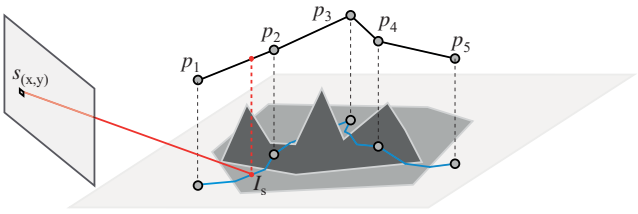


Figure 6: *Pixel back-projection and vector map location.*

Using the back-projected 3D point location I_s of a pixel $s_{(x,y)}$ with depth $d_{(x,y)}$ from the G-buffer we thus have determined the cluster $C_{u,v}$ containing any potential line candidates. We now have to determine if the point I_s lies within a certain distance of any line segment $l_i \in C_{u,v}$ in the 2D vector map plane. As illustrated in Fig. 6, point I_s lies inside a certain distance to the line segment p_1, p_2 of the vector map. Thus it is considered to be part of that line segment and the pixel $s_{(x,y)}$ can be colored accordingly using the current style and visualization parameters.

For large and complex vector maps as shown in Figs. 10(a), 10(c) and 10(f), however, per-pixel line identification and point-to-line distance tests can become costly in our deferred line shading approach and some further optimizations are called for as described below.

The point-to-line search and distance calculation within a cluster $C_{u,v}$ should be optimized to keep the per-pixel computation cost low. Fig. 10(d) shows a heatmap indicating the varying cost effort to find corresponding line segments. Given the varying number of line segments in different clusters, for large vector maps we organize the line segments within each cluster in a BVH as already mentioned in Sec. 3.1 to limit the number of distance test computations.

Given a pixel's position I_s in world coordinates and in vector-map space, the BVH can be traversed effectively to identify the leaf nodes containing any line segments $l_i' \subseteq C_{u,v}$ which are potentially closer than a certain given distance from I_s. Only for this subset of lines l_i' the point-to-line distance has eventually to be computed. Therefore, even for very large vector maps with many millions of lines, the amount of line distance tests required per cluster is eventually reduced to a small number and can thus be performed in a fragment shader for each pixel efficiently.

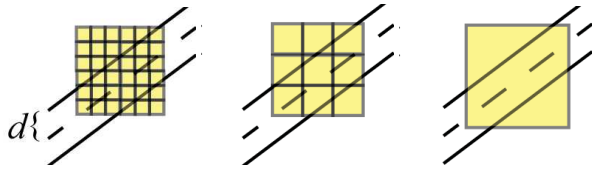
3.3 Line Styles and Antialiasing

The deferred line rendering method outlined above can further support visualization features beyond bare line rendering. In particular, the screen-space per-pixel shading allows the implementation of advanced colorization decisions like applying different line styles and line patterns on-the-fly during interactive rendering. Cross-sectional color patterns can easily be incorporated into a generic line shader taking the width of the line feature and the distance of the pixel to the line into account (see also Fig. 8 and results in the next section). Longitudinal procedural patterns can be applied using pattern buffers and more complex line shaders as well. Furthermore, given the pixel-to-line distance to the closest or all pixel-intersecting line features, alpha-blended compositing of multiple features or over background can be achieved. Dynamically varying or view-dependent visualization parameters can also be incorporated into the deferred line shading process, as demonstrated e.g. in Fig. 9 with an interactive lens function.

Thin lines can suffer from aliasing artifacts as shown in Fig. 10(c) where distant contour lines at the back become discontinuous. This effect may appear in several ways and is caused by the difference of the line size and the size of one pixel at this distance. In our manual GPU implementation, we distinguish three cases as described in Fig. 7. In the first, normal case when the line width covers several pixels, normal pixel drawing works and common artifacts can be reduced by supersampling or other standard antialiasing algorithms. The second case describes artifacts due to procedural patterns, see Fig. 7(b), which are not easily solved even when using alpha-blending techniques. We handle this case by reverting to the main color of the line pattern for line segments at far distances to convey the primary association to feature categories. The result can be seen in Fig. 10(b). The third case corresponds to a line segment missing the line test for some pixels due to the line width being smaller than the pixel at far distances. This is similar to rendering phone wires as described in [Persson 2012]. The main idea is to adjust the line width for distant points and color the pixel using alpha-blending based on the line's approximative coverage of the pixel. This is estimated from the ratio between the line's width and the pixel's extent.

4 Results

Our implementation runs on an Intel Core i7 3.5 GHz, 16 GB RAM, Nvidia GTX680 (4GB RAM, 1920×1080) machine with C++ and OpenGL. It allows us to load and interactively visualize large scale terrain and vector map datasets. In particular, the system supports exploration of large scale vector maps interactively in a full 3D environment. Tab. 2 lists the vector map datasets used in our tests.



(a) d > pixel (b) d ≈ pixel (c) d < pixel

Figure 7: *Three cases where aliasing artifacts can occur: (a) Regular line rendering staircase artifacts appear at distances where the line width d is bigger than the pixel size. (b) If d is almost equal to the pixel size artifacts may appear by wrong hits in the pattern buffers when using line-style patterns. (c) If d is smaller than the pixel size parts of a line segment are hidden and wrongly identified as background.*

Data set	Vector map Lines	Cluster max. size	Line Assignment	Render time
Vorarlberg (higher streets)	193,042	1,106	9.8 s	50 ms
Switzerland (TLM streets)	16,556,412	3,429	101.7 s	55 ms
Carinthia (isolines)	31,297,095	4,650	120.7 s	190 ms

Table 2: *Dataset overview. For each vector map we used a fixed cluster grid of* 256×256*. Average render times measured for viewpoints (3840 × 2160 for 2 × supersampling) shown in Fig. 10(b), 10(a), 10(c) including ∼15 ms for rendering terrain.*

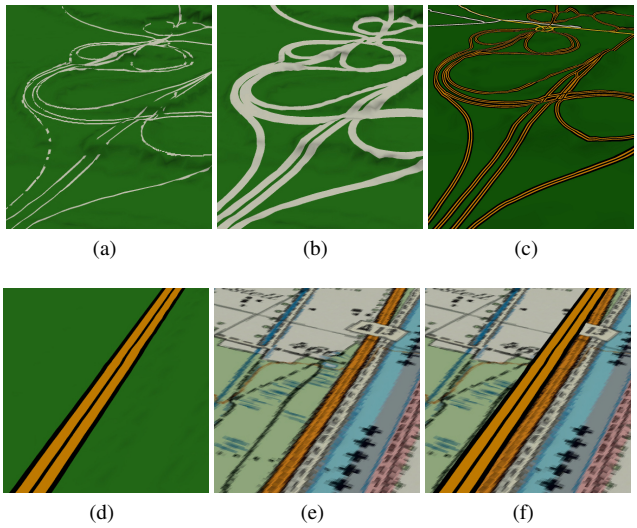
Experimental results show that our approach can be used for large scale interactive vector map visualization, see also Fig. 10. In comparison to texture-based visualizations, it can be guaranteed that the visual result is a pixel precise rendering as demonstrated in Fig. 8. Our system maintains full pixel precision at any zoom-in factor even near to the terrain. To achieve this with texture mapping it would require a much more complex and elaborate solution still suffering from resolution artifacts, see also Fig. 8(f). In contrast to geometric line rendering approaches, intersecting or floating line artifacts as shown in Fig. 2(a), as well as z-fighting problems as shown in Fig. 8(a) can be avoided.

Our deferred line rendering solution also efficiently supports interactively changing visualization parameters such as line size or styling properties as well as view-dependent data selection without reloading or re-rasterization of textures. A dynamic view-dependent vector map data-lens example is shown in Fig. 9.

A regular grid based clustered line buffer without additional hierarchical line segment organization is capable of handling vector maps of up to 200,000 line segments and 1,100 lines per cluster. The use of hierarchical spatial line indexing within each cluster further makes the interactive exploration of much larger vector maps possible. Vector maps with several millions of line segments can be visualized at interactive frame rates with our methods using the maximal cluster sizes indicated in Tab. 2. The rendering effort per pixel depends on the number of lines and their organization within each cluster in the line buffer. Fig. 4 shows an example for the content of the cluster information. Fig. 10(d) illustrates the per-pixel line search and distance test costs based on our clustered line buffer and local bounding volume line hierarchies.

5 Conclusion

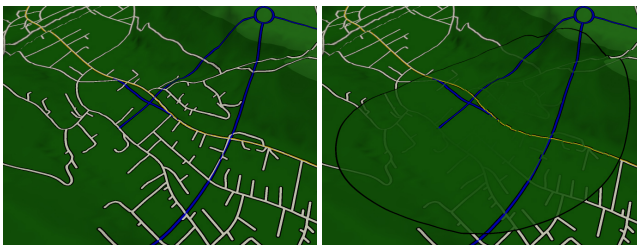
In this paper we present a novel approach for handling GIS vector data sets within interactive 3D environments. Our approach represents an efficient and flexible solution for different purposes. It can be used for interactive editing and adaptive or view-dependent



(a) (b) (c)

(d) (e) (f)

Figure 8: *Comparison of our approach (b,c,d) with normal 3D geometry (a) and texture based rendering (e,f): (a) A street rendered as a simple geometric object. (b) Simple line rendering with our method, and (c) applying some advanced vector styling. (d) A street rendered with our algorithm, and (e) using texture mapping at the highest resolution available for this texture set. (f) Overlay of (d) and (e) for direct comparison.*



(a) (b)

Figure 9: *Example for interactive editing functionality. It is possbile to interactivly fade out specific categories of streets from the vector map on a pixel precise basis using alpha blending.*

styling of vector maps as well as for large scale visualizations. In particular, it is also suitable for dynamic level-of-detail and out-of-core geographic visualization systems. Our approach relies on multiple rendering passes and was implemented using modern graphics hardware. However, as hardware is continuing to improve in terms of graphics functionality, this limitations may be overcome in the near future.

Our next steps are the extension of the system to work with polygonal objects as well as further improve rendering speed. In addition, future plans include an extensive evaluation and incorporating the system with a vector map out-of-core level-of-detail system to make larger data sets accessible.

Acknowledgements

The authors want to thank the Federal Office of Topography Swisstopo for providing the Swiss VECTOR25 and SwissTLM data sets as well as the Landesvermessungsamt Feldkirch, Austria, for providing the data sets of Vorarlberg.

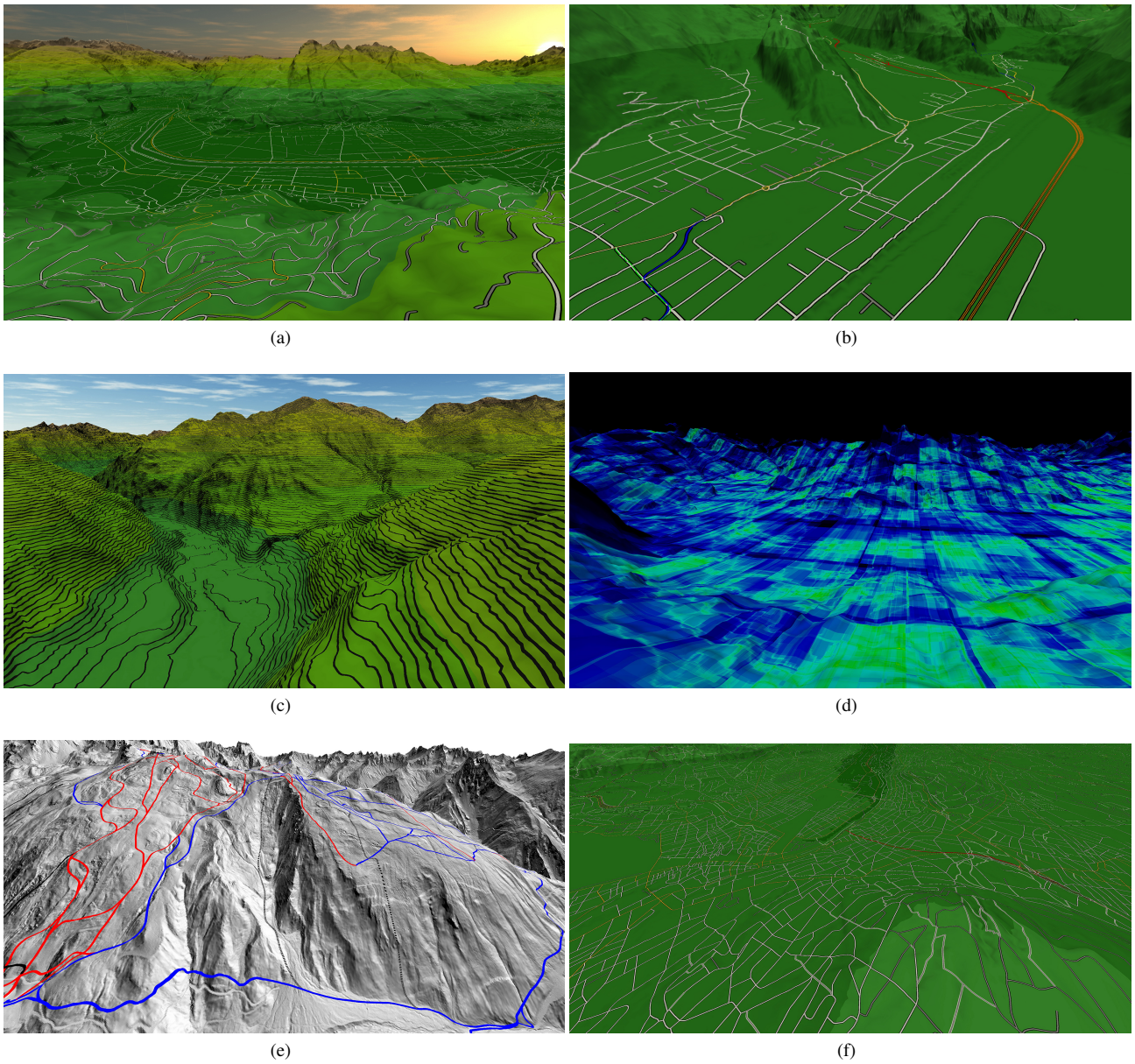


Figure 10: *(a) An example for line rendering of street data with styling pattern. (b) Example showing the smooth transition of a procedural pattern to solve alising artifacts shown in Fig. 7(b). (c) Carinthia isolines with black contour lines and distant lines suffering from aliasing artifacts. (d) A heatmap illustrating the bintree line search and intersection traversals cost from black (zero traversals) over blue to green. (e) Visualization of ski slopes over a hill-shade textured terrain with blue and red colored slopes for beginner and advanced levels respectively. Off-piste tracks are shown in a stippled black style pattern. (f) Swiss TLM street data containing 21 different categories of streets and using multiple line styles. Streets smaller than 4m wide using white and grey combinations, streets with 6m width are shown in yellow, and streets more than 10m wide are shown in red. Highways are highlighted in orange.*

References

BAVOIL, L., AND SAINZ, M. 2008. Screen space ambient occlusion. In *ShaderX 7*, NVIDIA Corporation.

BÖSCH, J., GOSWAMI, P., AND PAJAROLA, R. 2009. RASTeR: Simple and efficient terrain rendering on the GPU. In *Proceedings Eurographics Areas Papers, Scientific Visulization*, 35–42.

BRUNETON, E., AND NEYRET, F. 2008. Real-time rendering and editing of vector-based terrains. *Computer Graphics Forum 27*, 2 (April), 311–320.

CHAJDAS, M. G., MCGUIRE, M., AND LUEBKE, D. 2011. Subpixel Reconstruction Antialiasing for Deferred Shading. *Proceedings Interactive 3D Graphics and Games*, 15–22.

COZZI, P., AND RING, K. 2011. *3D Engine Design for Virtual Globes*. A. K. Peters, Ltd.

DAI, C., ZHANG, Y., AND YANG, J. 2008. Rendering 3D vector data using the theory of stencil shadow volumes. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 37*, 643–648.

DENG, B., XU, D., ZHANG, J., AND SONG, C. 2013. Visualization of vector data on global scale terrain. In *Proceedings International Conference on Computer Science and Electronics Engineering*, Atlantis Press, Advances in Intelligent Systems Research, 85–88.

DICK, C., KRÜGER, J., AND WESTERMANN, R. 2009. GPU ray-casting for scalable terrain rendering. In *Proceedings Eurographics Areas Papers*, 43–50.

GOSWAMI, P., MAKHINYA, M., BÖSCH, J., AND PAJAROLA, R. 2010. Scalable parallel out-of-core terrain rendering. In *Proceedings Eurographics Symposium on Parallel Graphics and Visualization*, 63–71.

KANG, H., JANG, H., CHO, C.-S., AND HAN, J. 2015. Multiresolution terrain rendering with GPU tessellation. *The Visual Computer 31*, 4 (April), 455–469.

KERSTING, O., AND DÖLLNER, J. 2002. Interactive 3D visualization of vector data in GIS. In *Proceedings ACM SIGSPATIAL International Conference on Advances in GIS*, 107–112.

LIKTOR, G., AND DACHSBACHER, C. 2012. Decoupled deferred shading for hardware rasterization. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 143–150.

LIVNY, Y., KOGAN, Z., AND EL-SANA, J. 2009. Seamless patches for GPU-based terrain rendering. *The Visual Computer 25*, 3 (February), 97–208.

LOSASSO, F., AND HOPPE, H. 2004. Geometry clipmaps: Terrain rendering using nested regular grids. *ACM Transactions on Graphics 23*, 3 (August), 769–776.

MAHDAVI-AMIRI, A., ALDERSON, T., AND SAMAVATI, F. 2015. A survey of digital earth. *Computers and Graphics 53* (December), 95–117.

OHLARIK, D., AND COZZI, P. 2011. A screen-space approach to rendering polylines on terrain. In *ACM SIGGRAPH Posters*, 68:1–1.

OLSSON, O., BILLETER, M., AND ASSARSSON, U. 2012. Clustered deferred and forward shading. In *Proceedings ACM SIGGRAPH/Eurographics Symposium on High-Performance Graphics*, 87–96.

PERSSON, E., 2012. Graphics gems for games: Findings from avalanche studios. ACM SIGGRAPH Advances in Real-Time Rendering in Games - Course Material, August.

RIPOLLES, O., RAMOS, F., PUIG-CENTELLES, A., AND CHOVER, M. 2012. Real-time tessellation of terrain on graphics hardware. *Computers & Geosciences 41* (April), 147–155.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible rendering of 3-D shapes. In *Proceedings ACM SIGGRAPH*, 197–206.

SCHNEIDER, M., GUTHE, M., AND KLEIN, R. 2005. Real-time rendering of complex vector data on 3D terrain models. In *Proceedings International Conference on Virtual Systems and Multimedia*, 573–582.

SUN, M., LV, G. L., AND LEI, C. 2008. Large-scale vector data displaying for interactive manipulation in 3D landscape map. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 37*, 507–512.

VAARANIEMI, M., TREIB, M., AND WESTERMANN, R. 2011. High-quality cartographic roads on high-resolution DEMS. *Journal of Winter School of Computer Graphics 19*, 41–48.

WANG, X., LIU, J., AND BI, J. 2009. Rendering of vector data on 3D virtual landscapes. In *Proceedings IEEE International Conference on Information Science and Engineering*, 2125–2128.

WARTELL, Z., KANG, E., WASILEWSKI, T., RIBARSKY, W., AND FAUST, N. 2003. Rendering vector data over global, multi-resolution 3D terrain. In *Proceedings Eurographics Symposium on Data Visualization*, 213–222.

WILKIE, D., SEWALL, J., AND LIN, M. C. 2012. Transforming GIS data into functional road models for large-scale traffic simulation. *IEEE Transactions on Visualization and Computer Graphics 18*, 6, 890–901.

XU, Y., SUI, Z., WENG, J., AND JI, X. 2010. Visualization methods of vector data on a Digital Earth System. In *Proceedings International Conference on Geoinformatics*, 1–5.

YANG, L., ZHANG, L., MA, J., KANG, Z., ZHANG, L., AND LI, J. 2011. Efficient simplification of large vector maps rendered onto 3D landscapes. *IEEE Computer Graphics and Applications 31*, 2 (March/April), 14–23.